Universidad Zaragoza
1542

Trabajo Fin de Grado

Inteligencia Artificial aplicada al problema de scattering en una dimensión

Autor

Pablo Antonio Calvo Barlés

Directores

Luis Martín Moreno

David Zueco Láinez

FACULTAD DE CIENCIAS

2020/2021

Título del resumen

RESUMEN

En este trabajo mostramos cómo una red neuronal artificial llega a ser capaz de realizar cálculos de resolución de la ecuación de Schrödinger en un sistema de scattering unidimensional a través del aprendizaje sobre ejemplos de entrenamiento. Entre los modelos propuestos, algunos permiten calcular el espectro de transmisión a partir de un potencial de scattering dado, mientras que otros son capaces de resolver el problema inverso, infiriendo la forma del potencial a partir de un espectro.

Índice

Capítulo 1

Introducción y objetivos

El Aprendizaje Automático o *Machine Learning* es un campo de la Inteligencia Artificial que estudia las técnicas y algoritmos informáticos con los que una máquina puede aprender a hacer predicciones sobre conjuntos de datos. Este campo está en continuo avance como causa del amplio rango de aplicación que tiene en las distintas ramas del conocimiento. Además, se ha visto muy beneficiado por las de grandes cantidades de datos y la creciente potencia computacional de la que se dispone hoy en día.

Este conjunto de técnicas tienen un gran potencial para realizar cálculos de diversos tipos y complejidades, así como de realizar análisis estadísticos y predictivos. Es por esto que también se han abierto paso entre muchas ramas de la física, tales como la astronomía, física de partículas, materia condensada, biofísica, etc.

En este trabajo varios modelos de aprendizaje automático basados en redes neuronales artificiales son propuestos como herramientas de cálculo para la resolución de la ecuación de Schrödinger en geometría de scattering unidimensional. Para ello, los modelos han aprendido sobre conjuntos de datos que hemos generado artificialmente por ordenador. Por otro lado, hemos programado dichos modelos con el lenguaje de programación *Python* y utilizando la biblioteca de Código Abierto *Keras*.

En la primera mitad del trabajo, haremos una introducción a la teoría de redes neuronales con el fin de entender su funcionamiento. Después, revisaremos brevemente la teoría cuántica de scattering y plantearemos el problema en el que se basa el trabajo. En la segunda mitad, describiremos los modelos y mostraremos sus resultados.

Capítulo 2

Redes neuronales artificiales

En este primer capítulo de teoría vamos a explicar qué son y cómo funcionan las redes neuronales artificiales. Después, mostraremos cómo pueden aprender de unos datos de entrenamiento por medio de la minimización de la denominada función de coste y explicaremos los principales problemas que se pueden dar en el proceso de aprendizaje, junto con diversas técnicas para mejorarlo.

2.1. ¿Qué es una red neuronal?

Una red neuronal artificial puede describirse como una "caja negra" de procesamiento de información, ya que recibe un conjunto de datos de entrada o *inputs*, realiza operaciones sobre ellos y obtiene unos datos de salida u *outputs*. Lo interesante de esta técnica es que no somos nosotros quienes escribimos el algoritmo con el que se procesan los inputs, sino que es la propia red la que aprende cómo procesarlos a partir de unos datos I/O de referencia. A estos los llamamos *ejemplos de entrenamiento*.

Antes de tratar las redes neuronales en sí, es necesario aclarar cómo funciona una neurona artificial, que es la unidad mínima de procesamiento en una red. Cada neurona recibe un conjunto de números $\{x_i\}$ y calcula otro número a aplicando la siguiente serie de operaciones: primero, realiza una combinación lineal utilizando unos parámetros $\{w_i\}$ llamados *pesos*. Después, suma un valor b llamado *bias*, obteniendo el valor z. Por último, genera el número a aplicando una *función de activación* no lineal f al valor anterior (a también puede llamarse activación de la neurona). El proceso se puede visualizar en la figura 2.1 y en la siguiente expresión:

$$a = f(z) = f(\sum_i w_i x_i + b) \tag{2.1}$$

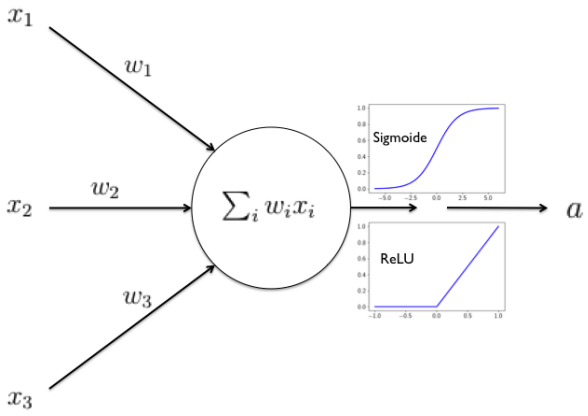

Figura 2.1: Representación de una neurona artificial.

Tanto los pesos como el bias son característicos de la neurona, a diferencia de $\{x_i\}$ y a. Por otro lado, las dos funciones de activación más utilizadas son la función *sigmoide* $((1+e^{-z})^{-1})$ y la función *ReLU* $(max(0,z))$, cuya forma también se muestra en la figura 2.1. Es fácil darse cuenta de que cuando $\sum_i w_i x_i$ supera el umbral $-b$, entonces $z > 0$ y las dos funciones crecen rápidamente o, como se dice en argot de redes neuronales inspirado por el comportamiento de nuestras neuronas, *se activan*.

Una vez claro el concepto de neurona artificial, construir una red consiste en conectar unas neuronas con otras, de manera que los *outputs* de unas sean los *inputs* de otras. Este tipo de estructura de procesamiento otorga una gran variedad de posibilidades de cálculo ya que, según el *teorema de aproximación universal* [1], cualquier función multivariable puede ser aproximada por una red neuronal artificial.

Existe un número enorme de arquitecturas de red posibles, así que vamos a considerar la más sencilla. La red neuronal de la figura 2.2 está formada por capas de neuronas *densas* (*dense layers*), lo que significa que cualquiera de las neuronas que compone una capa está conectada a todas las neuronas de la anterior. En esta red, además, podemos distinguir la *capa input*, que simplemente envía a la siguiente capa los datos que recibe la red (no tiene pesos ni bias), la *capa output*, que envía los *ouputs* de la red, y las *capas ocultas* (*hidden layers*), que son las que están en medio. Las redes que tienen dos o más capas ocultas se llaman *Redes Neuronal Profundas* (*Deep Neural Networks, DNNs*).

El conjunto de todos los pesos y *biases* de las neuronas constituyen los parámetros de la red y se expresan con la siguiente notación: por un lado, w^l_{jk} es el peso de la neurona j en la capa l que conecta con la neurona k en la capa $l-1$. Por otro, b^l_j es el

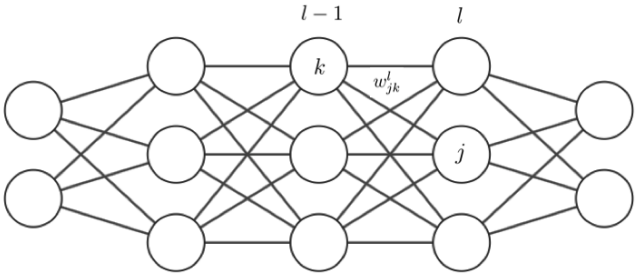

Figura 2.2: Visualización de red neuronal artificial densa.

bias de la neurona j en la capa l (ver 2.2). Además, los datos *input* de la red se denotan con el vector $\vec{x} \in \mathbb{R}^n$, siendo n el número de neuronas de la primera capa, mientras que las activaciones de una capa l se denotan con $\vec{a}^l = (a^l_1, ..., a^l_j, ...a^l_m)$, siendo m el número de neuronas de la capa en cuestión. El *output* o *predicción* de la red es denotado con $\vec{a}$ y, si la última capa tiene índice L, entonces $\vec{a} = \vec{a}^L$.

2.2. Aprendizaje automático: el descenso del gradiente

Para el proceso de entrenamiento vamos a proporcionar a la red un conjunto de ejemplos, que consisten en un número N de datos de entrada $\{\vec{x_1}, ..., \vec{x_N}\}$ junto con los respectivos datos de salida $\{\vec{y_1}, ..., \vec{y_N}\}$. Estos últimos son los que la red calcularía (a partir de los primeros) en el caso de que esté actuando correctamente, por lo que son llamados *outputs deseados*. El objetivo del aprendizaje es, entonces, encontrar los parámetros de red tales que las predicciones $\{\vec{a}\}$ se aproximen lo máximo posible a dichos *outputs*. Definimos la *función de coste* cuadrática como

$$C(w, b) = \frac{1}{2N} \sum_{i=1}^{N} ||\vec{y_i} - \vec{a}(\vec{x_i})||^2, \tag{2.2}$$

donde (w, b) representa $(\{w^l_{jk}\}, \{b^l_j\})$ por simplicidad y $|| \cdot ||$ es el operador de norma euclídea. Por otra parte, definimos el coste sobre un ejemplo i como $C_i = \frac{1}{2}||\vec{y_i} - \vec{a}(\vec{x_i})||^2$, de forma que la función de coste viene dada por el promedio de estos. En efecto, $\vec{a}$ es función de los parámetros de la red y, por lo tanto, C también lo es. Además, es una función no negativa y alcanza el valor mínimo $C = 0$ cuando $\vec{a}(\vec{x_i}) = \vec{y_i}\ \forall i$. Para optimizar los parámetros (w, b) hasta llegar a este mínimo global, es muy común utilizar el *algoritmo del descenso del gradiente*, que procederemos a explicar en las siguientes líneas. Sin embargo, existen muchos otros algoritmos u

optimizadores que realizan esta tarea, por ejemplo *RMSprop* o *Adam*.

Vamos a suponer que estamos en un punto cualquiera $\nu \equiv (w, b)$ del espacio de parámetros y nos movemos al punto ν' con una variación arbitraria muy pequeña $\Delta\nu = (\Delta w, \Delta b)$. Al hacer esto se produce un cambio ΔC en la función de coste, que se puede calcular de forma aproximada como:

$$\Delta C \approx \nabla C \cdot \Delta\nu, \tag{2.3}$$

siendo ∇ el operador gradiente respecto de los parámetros (w, b). Dado que queremos minimizar la función de coste, buscamos el vector $\Delta\nu$ tal que ΔC sea lo más negativo posible. Esto se consigue con $\Delta\nu = -\eta\nabla C$, de manera que $\Delta C \approx -\eta||\nabla C||^2$. El parámetro η, que se conoce como *learning rate*, tiene que ser positivo para que se cumpla $\Delta C < 0$, además de suficiente pequeño para asegurarnos de que la aproximación 2.3 sea válida.

Una vez dado el paso de ν a ν', repetimos el proceso de forma iterativa para minimizar la función de coste hasta aproximarnos al mínimo global. Así, actualizamos la posición en el espacio de parámetros utilizando la regla $\nu \to \nu' = \nu - \eta\nabla C$. Si expandimos la notación componente a componente, obtenemos:

$$w^l_{jk} \to w^l_{jk}{}' = w^l_{jk} - \eta\frac{\partial C}{\partial w^l_{jk}} \tag{2.4}$$

$$b^l_j \to b^{l\prime}_j = b^l_j - \eta\frac{\partial C}{\partial b^l_j} \tag{2.5}$$

2.3. Problemas en el aprendizaje y técnicas de mejora

2.3.1. Backpropagation

Está claro que para cada actualización de los parámetros es necesario obtener el valor de las derivadas parciales del coste C con respecto a los pesos y *biases*. Aquí nos encontramos con un inconveniente, debido a que el número de parámetros y de ejemplos que hay en una red es enorme. Hacer este cálculo de forma analítica sería inviable, y obtener las derivadas numéricamente [1] sería muy costoso computacionalmente. Por poner un ejemplo, en este trabajo hemos entrenado redes con un número de

[1] $\frac{\partial C}{\partial \theta} \approx \frac{C(\theta+\Delta\theta)-C(\theta)}{\Delta\theta}$, siendo θ un parámetro de la red.

parámetros del orden de 10^5 y número de ejemplos de 10^4.

Para solventar el problema del número de parámetros, utilizamos el *algoritmo de backpropagation*. Lo primero que hacemos es definir la cantidad $\delta_j^l = \frac{\partial C}{\partial z_j^l}$ ($\vec{\delta}^l$ de forma vectorial) como una medida del error de la neurona j en la capa l, siendo z_j^l el *output pesado* antes de aplicar la función de activación. Teniendo esto en cuenta, para cada ejemplo i realizamos los siguientes pasos:

1. A partir de $\vec{x_i}$, generamos los *outputs* pesados $\vec{z}^l$ y activaciones $\vec{a}^l$ desde la capa $l = 1$ hasta $l = L$ (*feedforward*).

2. Calculamos los errores de la última capa: $\delta_j^L = \frac{\partial C_i}{\partial a_j^L}\sigma'(z_j^L)$.

3. Empezando por $\vec{\delta}^L$, calculamos iterativamente los errores de todas las capas ($l = L-1, ..., 1$) con la siguiente expresión: $\delta_j^l = \sum_k w_{kj}^{l+1}\delta_k^{l+1}\sigma'(z_j^l)$. De este paso viene el nombre *backpropagation*, ya que propagamos los errores de las neuronas hacia atrás.

4. Una vez obtenidas las activaciones y errores, ya podemos calcular las derivadas parciales de C_i con respecto a los pesos, $\frac{\partial C_i}{\partial w_{jk}^l} = a_k^{l-1}\delta_l^j$, y a los biases, $\frac{\partial C_i}{\partial b_j^l} = \delta_j^l$.

Al final, calculamos ∇C promediando sobre todos los ejemplos. Las cuatro expresiones introducidas en estos pasos constituyen las ecuaciones de *backpropagation* y se pueden deducir utilizando la regla de la cadena en derivadas parciales [1]. Este algoritmo ha resultado ser muy eficiente computacionalmente, por lo que su uso está muy generalizado.

2.3.2. Stochastic gradient descent

Otra técnica comúnmente muy utilizada para mejorar la eficiencia del aprendizaje es el *stochastic gradient descent* (SGD). Consiste en aproximar ∇C sumando sobre un conjunto reducido de m ejemplos escogidos aleatoriamente, en lugar de sumar sobre el total de N. A este subconjunto de ejemplos lo llamamos *mini-batch*. Tras hacer una actualización con el primero, se escoge el siguiente *mini-batch* entre el resto de ejemplos que quedan, y así sucesivamente.

Diremos que hemos completado una *época de entrenamiento* cuando hayamos utilizado los N ejemplos en grupos de *mini-batches*. Para monitorizar el aprendizaje sobre un conjunto de datos, definimos la *curva de entrenamiento* como la gráfica que

representa el valor de C en función del número de épocas.

Gracias a la técnica del SGD conseguimos dos cosas: por un lado, el cálculo de las derivadas es mucho más eficiente. Por otro lado, añadimos una componente estocástica a la trayectoria del descenso del gradiente, ya que en cada actualización se cambian los ejemplos del *mini-batch* de forma aleatoria. Esto último es muy útil para evitar que los parámetros se queden estancados en un mínimo local de $C(w, b)$.

2.3.3. Overfitting y elección de hiperparámetros

Un problema importante después del entrenamiento de una red se encuentra en su capacidad de generalización, es decir, en cómo hace predicciones sobre datos que no forman parte de los ejemplos de entrenamiento. Vamos a considerar una red que ha minimizado con éxito la función de coste y, por tanto, hace buenas predicciones sobre los datos de entrenamiento. Para comprobar su nivel de generalización, reservamos una selección de ejemplos con los que la red no ha entrenado, a los que llamamos *datos de validación*. Después, observamos la curva de entrenamiento evaluada sobre estos, junto con la curva evaluada sobre los datos de entrenamiento. Generalmente, dichas curvas suelen minimizarse igual que en el ejemplo de la figura 2.3.

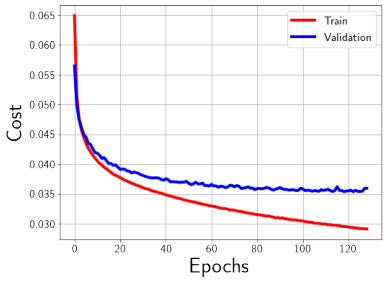

Figura 2.3: Ejemplo típico de curvas de entrenamiento y validación de un modelo de red neuronal.

Si, a diferencia de los datos de entrenamiento, la curva deja de disminuir o aumenta a partir de una cierta época, entonces probablemente el modelo tenga más parámetros de los necesarios. En este caso decimos que hay *overfitting*. Veamos un ejemplo sencillo que aparece en la figura 2.4 y que ilustra muy bien este fenómeno: se trata de conjunto de puntos $\{(x, y)\}$ de entrenamiento sobre los que queremos hacer un ajuste polinomial, y otro conjunto de datos de validación con los que comprobamos la capacidad de generalización del ajuste. Si lo hacemos con un polinomio de grado 8, el error con los ejemplos de entrenamiento es nulo, mientras que con los ejemplos de validación es muy alto. En ese caso hay *overfitting*. Sin embargo, cuando reducimos el número de

parámetros del modelo con un polinomio de grado 1, la predicción sobre los datos de validación mejora considerablemente y, en consecuencia, la capacidad de generalización.

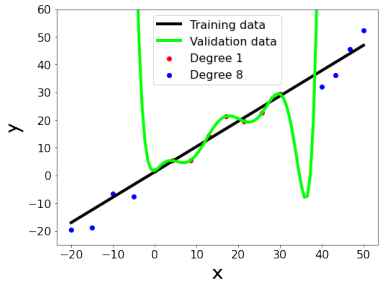

Figura 2.4: Ajuste polinomial.

Este problema ocurre muy habitualmente en las redes neuronales y hay muchas maneras de abordarlo. Lo primero que se puede hacer es reducir la complejidad del modelo (número de neuronas, número de capas, etc). Otra opción se conoce como *Early Stopping*, en la que detenemos el entrenaminto cuando la curva de validación deja de disminuir. Existen más técnicas, como el *Dropout*, *Regularización L1* y *L2*, etc, de las que no vamos a dar más detalles, pero están explicadas con mayor profundidad en el capítulo 3 del libro de Michael A. Nielsen [1].

Definimos como *hiperparámetros* al conjunto de elementos del modelo de red que están fijos durante el entrenamiento, es decir, al número da capas, número de neuronas por capa, *learning rate* o tamaño del *mini-batch*. En definitiva, monitorizar los datos de validación nos sirve para hacer la elección de dichos *hiperparámetros*. El objetivo es que la complejidad del modelo sea suficiente como para garantizar el aprendizaje y la disminución de la curva de entrenamiento, pero tampoco excesiva como para provocar *overfitting*.

Capítulo 3

Teoría cuántica de scattering

En el tercer capítulo vamos a establecer el contexto sobre la teoría cuántica en la que hemos basado este trabajo, así como explicar el problema físico que intentamos que una red neuronal resuelva y definir algunos elementos que son convenientes para abordarlo.

Consideramos una partícula cuántica en una dimensión en presencia de un potencial $V(x)$ que es nulo en todo el espacio excepto en $x \in [-l, l]$, donde tiene una forma arbitraria ($l = 1$ para el resto del trabajo). Sabemos que las funciones de onda estacionarias $\psi(x)$ con energía asociada E satisfecen la ecuación de Schrödinger independiente del tiempo:

$$-\frac{\hbar^2}{2m}\frac{d^2\psi}{dx^2} + V(x)\psi(x) = E\psi(x) \tag{3.1}$$

Las soluciones de onda libre en el dominio $x \notin [-l, l]$ vienen dadas por una combinación lineal de las funciones e^{ikx} y e^{-ikx}, siendo $k = \hbar^{-1}\sqrt{2mE}$ el número de ondas. Estas representan estados con momento lineal bien definido viajando hacia el sentido positivo y negativo del eje OX respectivamente. Para $x < -l$ se cumple

$$\psi(x) = Ae^{ikx} + A'e^{-ikx}, \tag{3.2}$$

mientras que para $x > l$ tenemos

$$\psi(x) = \tilde{A}e^{ikx} + \tilde{A}'e^{-ikx}. \tag{3.3}$$

La *matriz de transmisión* [2] $\mathbf{M}$ relaciona las amplitudes de onda A y A' en $x < -l$ con las amplitudes $\tilde{A}$ y $\tilde{A}'$ en $x > l$ de la siguiente manera:

$$\begin{pmatrix} \tilde{A} \\ \tilde{A}' \end{pmatrix} = \mathbf{M}\begin{pmatrix} A \\ A' \end{pmatrix}, \quad \mathbf{M} = \begin{pmatrix} M_{11} & M_{12} \\ M_{21} & M_{22} \end{pmatrix}, \tag{3.4}$$

donde los valores de esta matriz dependen del parámetro k y de la forma del potencial $V(x)$. Para potenciales en los que hay N barreras o deltas de Dirac concatenadas, la matriz de transmisión se puede expresar como el producto de las matrices de cada bloque de potencial por separado [3]:

$$\mathbf{M} = \mathbf{M}_N \mathbf{M}_{N-1} ... \mathbf{M}_2 \mathbf{M}_1 \tag{3.5}$$

En el problema que hemos tratado en este trabajo vamos a suponer que la partícula se acerca desde la región negativa del eje OX, lo que nos permite imponer la siguiente condición: $\tilde{A}' = 0$. Además, si tomamos $A = 1$, podemos definir los *coeficientes de transmisión y reflexión*, t y r, como las amplitudes complejas de la onda transmitida en $x > l$ y de la onda reflejada en $x < -l$. Teniendo en cuenta las condiciones impuestas y la definición 3.4, se escriben de la siguiente manera:

$$t = M_{11} - \frac{M_{21}}{M_{22}} M_{12} \tag{3.6}$$

$$r = -\frac{M_{21}}{M_{22}} \tag{3.7}$$

A continuación, definimos las cantidades conocidas como *transmitancia* T y *reflectancia* R, que vienen dadas por:

$$T = |t|^2 \tag{3.8}$$

$$R = |r|^2 \tag{3.9}$$

Estas magnitudes son una representación de la probabilidad de transmisión y reflexión de la partícula en el proceso de scattering bajo el potencial V(x). Es importante darse cuenta de que, al estar considerando estados estacionarios, la corriente de probabilidad $J(x)$[1] se conserva. De este hecho se deduce que $T + R = 1$ y que la matriz de transmisión cumple $\det \mathbf{M} = 1$. Además, T y R salen igual tanto si la partícula viene desde la derecha como desde la izquierda, por lo que el sistema presenta simetría bajo inversión espacial [2].

Al igual que ocurre con la matriz de transmisión, la transmitancia y reflectancia dependen de la energía de la partícula y la forma del potencial. Por lo tanto, es posible determinar una función de T en función del parámetro k que esté asociada a un potencial. A esta función la llamamos *espectro de transmisión* $T(k)$. Dado que este objeto es de gran importancia en el problema de scattering cuántico, el objetivo principal en el que hemos basado este trabajo es, dado un potencial arbitrario V(x), determinar cuál es el espectro de transmisión y viceversa.

[1] $J(x) = \frac{\hbar}{2mi}[\psi^*(x)\frac{d\psi}{dx} - \psi(x)\frac{d^2\psi^*}{dx^2}]$, ver [2]

Capítulo 4

Resultados

En este capítulo vamos a mostrar la aplicación de las redes neuronales al problema de scattering unidimensional en sus distintas variantes. Para ello, propondremos diferentes modelos de redes y presentaremos los resultados que obtienen tras haber sido entrenados.

4.1. Modelos de red y ejemplos de entrenamiento

Entre los modelos propuestos, distinguimos dos tipos:

1. *Modelos directos*, en los que la red recibe datos de un potencial $V(x)$ como *input* y obtiene transmitancias como *output*. Lo hemos realizado de dos maneras: en un caso (*NNspectrum*), la red calcula un espectro discreto $T(k)$ completo, de forma que cada neurona de la última capa devuelve una transmitancia $T(k_n)$, desde k_{min} hasta k_{max} con un intervalo de discretización Δk (los discretizados en este trabajo son de 300 valores de k). En el otro (*NN1T*), la red recibe el número k de la partícula junto con los datos del potencial y calcula un único valor de transmitancia. Por tanto, la capa *output* tiene una neurona. Para obtener el espectro completo con este último, el modelo tiene que realizar una predicción de T por cada valor de k por separado.

2. *Modelos inversos*, en los que la red recibe el espectro de transmisión discreto $T(k)$ como *input* y devuelve los datos del potencial $V(x)$ asociado.

Cualquiera de los modelos cuenta con activaciones de tipo sigmoide en las neuronas de la última capa. Dado que $0 < \sigma(z) < 1$, la función se adapta muy bien a los *outputs* en los modelos directos, ya que representan transmitancias y, por definición (3.8), están acotadas por ese intervalo. Para los modelos inversos, los datos respectivos al potencial pueden ser normalizados de tal forma que el rango de la función sigmoide cubra todos los posibles valores.

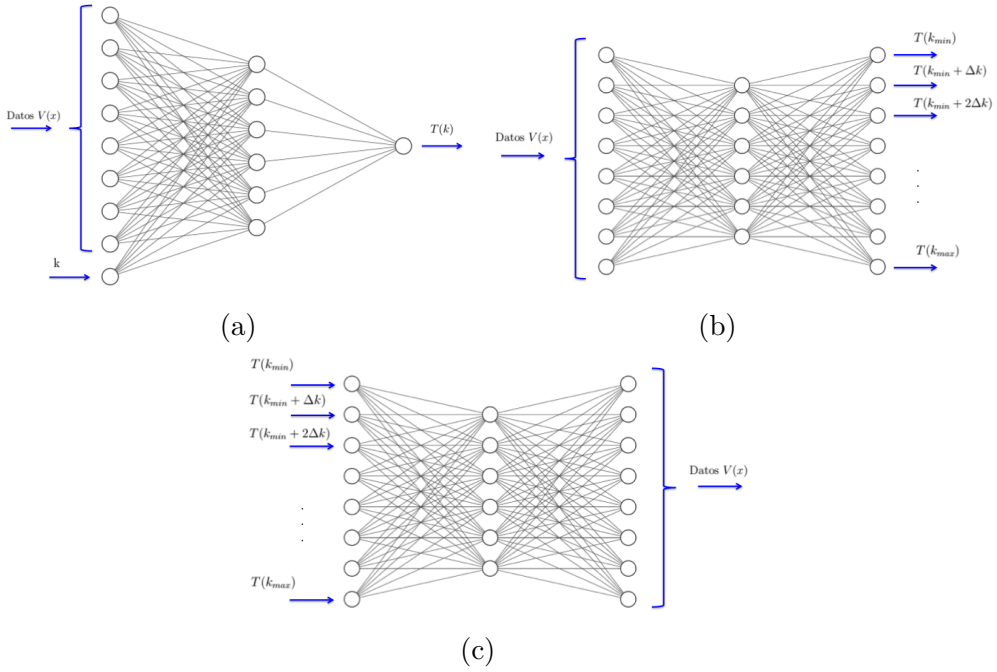

Figura 4.1: (a) Modelo directo *NN1T*. (b) Modelo directo *NNspectrum*. (c) Modelo inverso.

Los ejemplos de entrenamiento y validación consisten en parejas potencial-transmitancias: los potenciales se han generado de manera aleatoria y las transmitancias se han calculado utilizando métodos tanto analíticos como numéricos. Además, los datos que hemos generado tienen la forma que requiere cada tipo de modelo, ya sea como en (a), (b) o (c) en la figura 4.1.

En principio, los datos de un potencial podrían ser codificados con una discretización espacial, de forma similar a la discretización del espectro. Cada neurona de la capa *input* u *output* representaría un valor de V en un punto discreto x_n. Sin embago, en las próximas secciones veremos ejemplos concretos en los que los potenciales, a la hora de su generación, son parametrizados por un conjunto de variables mucho más reducido, las cuales tienen distinto significado dependiendo del tipo de potencial. Por esta razón podemos caracterizar $V(x)$ con muchas menos neuronas, ya que no es necesario introducir la discretización completa. Así, en definitiva, disminuimos el número de parámetros de red y la complejidad del modelo.

Nuestro objetivo ha sido que los modelos tengan la máxima capacidad de generalización posible, lo que significa que puedan hacer predicciones sobre una amplia variedad de formas de $V(x)$ y $T(k)$ con la máxima precisión. Esto lo hemos tenido en cuenta a la hora de generar los datos de entrenamiento y validación, por lo que hemos

valorado diferentes opciones que procedemos a presentar en las dos siguientes secciones.

4.2. Potencial de polinomios de Chebyshev

Consideramos potenciales $V(x)$ dados por la combinación lineal de n polinomios de *Chebyshev*, que son ortogonales[1] entre sí: $V(x) = \sum_{i=0}^{n-1} a_i T_i(x)$. Estos potenciales quedan completamente caracterizados por los coeficientes $\{a_i\}$ y por el valor máximo del potencial V_0. Por tanto, los ejemplos *input* de una red directa cuentan con $n+1$ números en el caso *NNspectrum* y $n+2$ en el caso *NN1T*.

Los potenciales han sido generados en *Python* con funciones del módulo *numpy.polynomial*. Además, los hemos multiplicado por un factor $e^{-\beta x^2/2}$ (con $\beta = 10$) para que decaigan suavemente. Por otra parte, para generar las transmitancias y espectros de los ejemplos *output* hemos utilizado dos métodos de resolución numérica de la ecuación de Schrödinger (ver anexo A).

La elección de este tipo de polinomios ortogonales es debida a la gran variedad de formas de $V(x)$ que se pueden llegar a generar con una combinación de unos pocos polinomios, además de por la cantidad de máximos y mínimos locales que se crean (ver ejemplo en la figura 4.2).

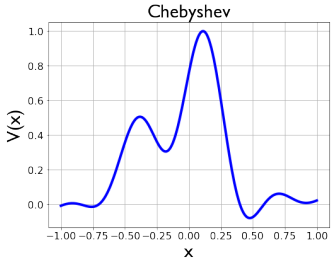

Figura 4.2: Ejemplo de potencial con $n = 10$ polinomios de *Chebyshev*.

A continuación mostramos las pruebas que hemos realizado con los dos modelos directos de red neuronal densa, *NN1T* y *NNspectrum*. En ambos casos hemos generado potenciales con $n = 10$ polinomios de *Chebyshev*, con $a_i \in [-1, 1]$ $\forall i$ y $V_0 \in (0, 4)$. Además, hemos minimizado el coste con el optimizador *RMSprop*. La arquitectura de capas ocultas de *NN1T* viene dada por[2]:

[50 (tanh), Dropout (0,15), 50 (σ), Dropout(0,15), 50 (σ)].

[1] Ortogonalidad bajo el producto escalar del espacio de Hilbert de funciones de cuadrado integrable.

[2] La primera capa tiene una activación de tipo tangente hiperbólica, que tiene forma similar a la sigmoide pero con rango de -1 a 1.

La red se ha entrenado durante 300 épocas con 7000 ejemplos de entrenamiento y 2000 de validación, y el resto de los hiperparámetros son: $\eta = 1{,}0$ y 5 ejemplos por *mini-batch*. Por otro lado, la arquitectura de *NNspectrum* es:

$$[100\ (\sigma),\ 100\ (\sigma)].$$

Esta red se ha entrenado durante 200 épocas con 8000 ejemplos de entrenamiento y 2000 de validación, mientras que el resto de hiperparámetros son: $\eta = 1{,}0$ y 5 ejemplos por *mini-batch*. Los resultados del entrenamiento se muestran en la figura 4.3.

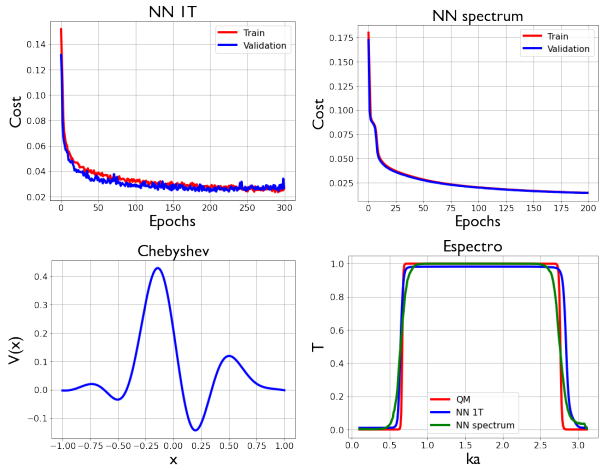

Figura 4.3: Resultados del entrenamiento. Las dos gráficas de arriba muestran las curvas de entrenamiento de *NN1T* y *NNspectrum* respectivamente. Abajo a la izquierda aparece un ejemplo de potencial proveniente de los datos de validación para *NNspectrum*. Por último, en la gráfica de la derecha se presentan las predicciones del espectro asociado al potencial. T está en función de ka, siendo a el intervalo de discretización utilizado en los métodos numéricos. *QM* indica el *output* deseado.

A la hora de calcular espectros de transmisión para los datos de entrenamiento y validación, encontramos otro problema. Hemos observado que, de manera general, únicamente se obtienen transmitancias iguales a 0 o 1, independientemente de $V(x)$ o de k. Sin opción intermedia, la partícula se transmite o se refleja completamente en casi todos los escenarios posibles. Este tipo de espectros, además de no tener mucho interés físico, están más determinados por el valor máximo del potencial que por su propia forma: en muchos casos, la transmitancia de la partícula aumenta bruscamente de 0 a 1 cuando su energía supera la barrera del potencial, como se puede apreciar en la figura 4.3. Con esta perspectiva, las redes neuronales han aprendido más bien a detectar el máximo de los potenciales que a calcular espectros dados por la resolución de la ecuación de Schrödinger en la geometría de scattering.

Aun así, el aprendizaje ha dado buenos resultados. Además, estos potenciales nos han servido como primer acercamiento al problema de este trabajo y han motivado la

búsqueda de otros tipos de potencial que dan lugar a espectros de mayor interés, como veremos en la sección que viene a continuación.

4.3. Potencial de Kronig-Penney

Esta vez vamos a considerar potenciales del modelo de Kronig-Penney [4] formados por varios bloques concatenados, cada uno con un potencial de delta de Dirac (δ) seguido de un intervalo de potencial nulo a:

$$V(x) = \sum_{j=0}^{N-1} \alpha_j \delta(x + l - ja), \tag{4.1}$$

con $x \in [-l, l]$ ($l = 1$) y siendo α_j el acoplamiento asociado a la delta de Dirac en la posición j-ésima. Para una partícula con número de ondas k, la expresión de la matriz de transmisión de un bloque j viene dada por [3]:

$$\mathbf{M}_j = \begin{pmatrix} (1 - \frac{i}{ka_j})e^{ika} & -\frac{i}{ka_j}e^{ika} \\ \frac{i}{ka_j}e^{-ika} & (1 + \frac{i}{ka_j})e^{-ika} \end{pmatrix}, \tag{4.2}$$

siendo $a_j = \hbar^2/m\alpha_j$. Vemos que, según la propiedad 3.5, la matriz de una cadena de deltas se obtiene como el producto de las matrices $\mathbf{M}_j$. Después, utilizando 3.6 y 3.8, podemos calcular de forma analítica la transmitancia asociada a un potencial de deltas de Dirac.

Es fácil darse cuenta de que, dada una distancia fija a, cualquier potencial con N deltas está completamente caracterizado por sus acoplamientos $\{\alpha_j\}$. Como consecuencia, hemos utilizado estos números como datos del potencial para la red. Además, los datos de entrenamiento y validación para los modelos propuestos en esta sección se han generado bajo las siguientes condiciones: $\hbar = m = 1$, $\alpha_j \in (0, 5]$ $\forall j$ y $k \in (0, 10]$.

La principal ventaja que muestran estos potenciales con respecto a los de *Chebyshev* se aprecia en los espectros de las figuras de las siguientes páginas. Claramente la complejidad de estos es mucho mayor y su forma aporta más información sobre los potenciales, así que consideramos que son de mayor interés para este trabajo.

Antes de presentar los modelos, definimos *curva de complejidad* (*Model Complexity, MC*) como la gráfica que representa la evaluación del coste sobre un conjunto de datos (tanto de ejemplo como de validación) en función del valor de un hiperparámetro, como el número de capas o el número de neuronas por capa. Hemos utilizado este

elemento con el fin de encontrar los hiperparámetros que optimicen la precisión de las predicciones y que eviten que se produzca *overfitting*. Cada punto de la curva corresponde a un modelo entrenado con un número de épocas determinado por la técnica de *Early Stopping*.

4.3.1. Modelos directos

Hemos construido los modelos directos con una arquitectura de capas ocultas densas y funciones de activación $ReLU$, mientras que la minimización del coste se ha realizado con el optimizador $Adam$[3]. En las pruebas realizadas, las redes se han entrenado con potenciales de un número fijo de deltas de Dirac.

Con el modelo *NN1T*, hemos optimizado por separado varias redes neuronales con potenciales de diferente número de deltas. De esta manera, hemos encontrado que las predicciones presentan fallos graves a partir de $N = 5$ deltas. En la figura 4.4 se muestran ejemplos de predicciones de espectros en cada uno de los casos, desde 2 a 6 deltas. Es fácil apreciar donde se encuentra el límite del modelo.

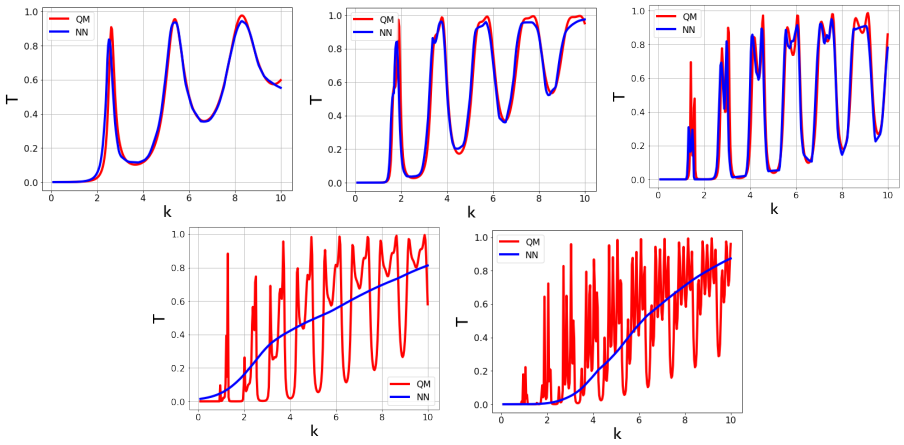

Figura 4.4: Ejemplos de predicción del modelo *NN1T*. Las gráficas de la fila de arriba corresponden a las redes entrenadas con 2, 3 y 4 deltas. Las de abajo, con 5 y 6.

Vamos a desarrollar cómo ha sido la optimización del modelo de potenciales con $N = 4$, que es el último que otorga buenos resultados. Para este caso, consideramos los siguientes hiperparámetros: 20000 ejemplos de entrenamiento, 4000 ejemplos de validación, $\eta = 10^{-3}$ y 20 ejemplos por *mini-batch*. Hemos realizado dos pruebas de complejidad: en la primera, fijamos una capa oculta y variamos el número de neuronas. En base al resultado de la figura 4.5, hemos fijado 220 neuronas. En la

[3] *Adam* es una extensión del SGD más efectiva computacionalmente.

segunda prueba, variamos el número de capas. Los resultados aparecen en la figura 4.6. Hemos observado que la precisión del modelo aumenta considerablemente de 1 a 4 capas ocultas. Podemos apreciar que, a lo largo de 140 épocas, la curva de entrenamiento se minimiza mucho más rápido en la red de 4 capas. Además, se presentan las predicciones sobre dos ejemplos de espectro. En ellos se observa que, con 4 capas, el modelo detecta los picos con mucha más precisión, además de hacerlo en un rango de k más extenso.

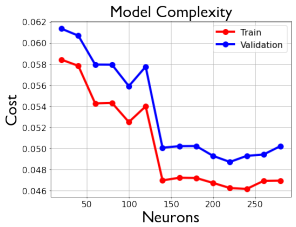

Figura 4.5: Curva de complejidad en función del número de neuronas para 1 capa oculta del modelo *NN1T*.

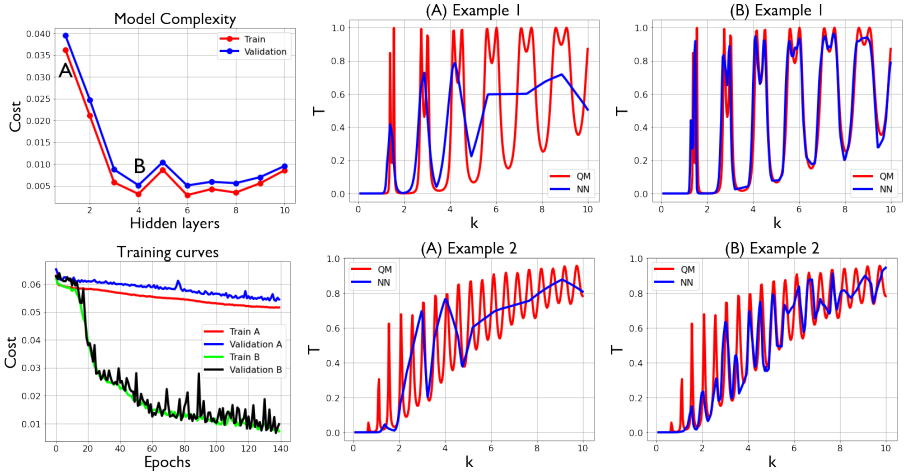

Figura 4.6: Curva de complejidad, curvas de entrenamiento y ejemplos de predicción del modelo directo *NN1T*. 'A' y 'B' etiquetan los modelos con 1 y 4 capas. Potencial asociado al ejemplo 1: $\{\alpha_i\} = \{3{,}6959101, 0{,}06020075, 0{,}15974319, 1{,}54157132\}$. Potencial asociado al ejemplo 2: $\{\alpha_i\} = \{3{,}1831926, 3{,}62941169, 3{,}76388581, 0{,}31334371\}$.

El modelo *NNspectrum* muestra mucha más facilidad de aprendizaje. En primer lugar, no hemos encontrado un límite de deltas de Dirac en el que el modelo muestre fallos de precisión. Por otro lado, las curvas de entrenamiento convergen en menos épocas y alcanzan valores más bajos. Vamos a mostrar los resultados del entrenamiento con potenciales de $N = 10$ deltas. Siguiendo el mismo procedimiento, hemos obtenido la curva de complejidad variando el número de neuronas y manteniendo una capa oculta fija. Después, hemos repetido el proceso con el número de capas (ver figura 4.7). Así, obtenemos el mejor modelo con 220 neuronas y 4 capas ocultas. El resto de hiperparámetros son: 8000 ejemplos de entrenamiento, 1990 ejemplos de validación, $\eta = 10^{-3}$ y 10 ejemplos por *mini-batch*.

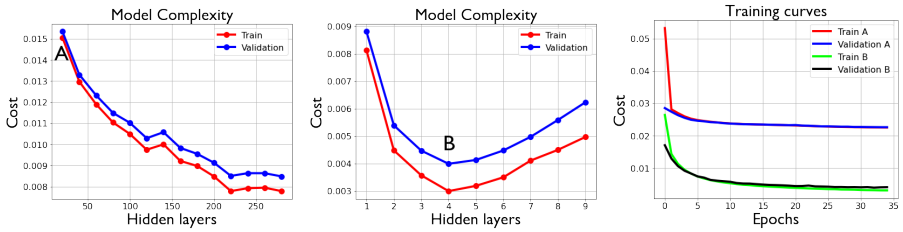

Figura 4.7: Curvas de complejidad en función del número de neuronas (izquierda), en función del número de capas (centro) y curvas de entrenamiento (derecha) en el modelo *NNspectrum*. 'A' etiqueta el modelo con 1 capa de 10 neruonas, y 'B' el de 4 capas de 220 neuronas.

La figura 4.8 muestra la comparación entre el mejor y el peor modelo según la evaluación del coste de validación. Estos se corresponden con los etiquetados como A y B en la figura 4.7. En efecto, la predicción del modelo B se ajusta mejor a los *outputs* deseados, pero aún así los dos son capaces de detectar los picos de los espectros.

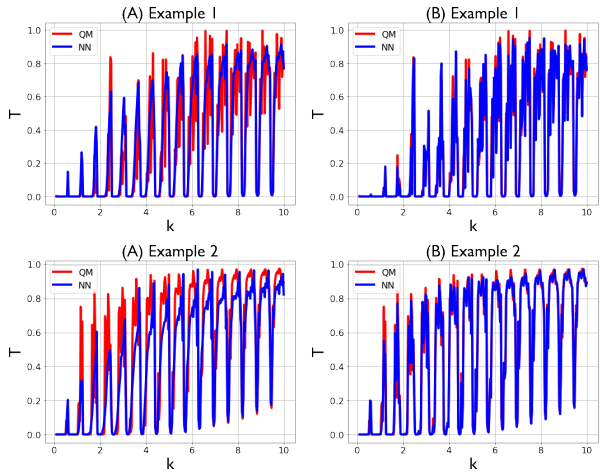

Figura 4.8: Ejemplos de predicción del modelo directo *NNspectrum*.

En definitiva remarcamos que ambos modelos, *NN1T* y *NNspectrum*, son capaces de calcular espectros de transmisión procedentes de potenciales de hasta 4 deltas de Dirac.

Para cadenas más largas, el modelo *NNspectrum* es claramente superior en cuanto a la precisión de las predicciones. Además, para una predicción similar, requiere de muchos menos parámetros de red y datos de entrenamiento que el modelo *NN1T*.

4.4. Modelos inversos

Procedemos a presentar en esta última sección el modelo inverso. Igual que en los directos, hemos utilizado activaciones $ReLU$ en las capas ocultas e implantado el optimizador *Adam*. Además, la red ha sido entrenada también con potenciales que compartan el mismo número de deltas. Por otra parte, los acoplamientos $\{\alpha_j\}$ de los potenciales de entrenamiento y validación han sido normalizados entre 0 y 1 para que el rango coincida con el de la función sigmoide.

Hemos visto en el capítulo 3 que el sistema cuántico que tratamos en este trabajo presenta una simetría bajo inversión espacial. Como consecuencia, un potencial $\{\alpha_0, \alpha_1, ..., \alpha_{N-2}, \alpha_{N-1}\}$ tendrá asociado el mismo espectro de transmisión que el potencial invertido $\{\alpha_{N-1}, \alpha_{N-2}, ..., \alpha_1, \alpha_0\}$. Supongamos un conjunto de potenciales de entrenamiento en el que hay dos que están invertidos entre sí y, por tanto, comparten el mismo espectro $T(k)$. En el entrenamiento, el optimizador minimizará el error de la predicción de $T(k)$ con ambos potenciales a la vez, así que el modelo inverso acabará mostrando fallos en las predicciones de ese espectro. Si esto ocurre con muchas parejas de potenciales, la precisión de la red se verá gravemente afectada. Para evitar este inconveniente, necesitamos generar datos dentro de un conjunto de potenciales tales que no haya dos invertidos entre sí. Para ello, hemos generado potenciales con acoplamientos uniformemente aleatorios bajo una condición: si $\alpha_0 < \alpha_{N-1}$ guardamos el potencial tal cual, mientras que si $\alpha_0 < \alpha_{N-1}$ guardamos el potencial invertido. El proceso se puede visualizar para potenciales de N=2 deltas en la figura 4.9.

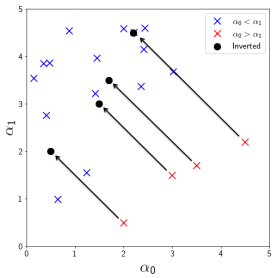

Figura 4.9: Representación de potenciales con N=2 deltas.

Vamos a mostrar los resultados del entrenamiento con potenciales de $N = 6$ deltas. De nuevo, hemos realizado pruebas con curvas de complejidad para el número

de neuronas por capa y el numero de capas ocultas (ver 4.10). Comenzando con la primera, hemos fijado 8000 datos de entrenamiento y 4000 de validación, mientras que los hiperparámetros son: 1 capa oculta, $\eta = 2 \times 10^{-4}$ y 20 ejemplos por *mini-batch*. Como es de esperar, hemos observado que la curva disminuye hasta que se satura en 200 neuronas, por lo que hemos fijado dicho hiperparámetro. Observemos ahora la curva de complejidad del número de capas. Esta vez, para conseguir mayor precisión en las predicciones, hemos aumentado el número de datos de entrenamiento a 14000. Además, para evitar fluctuaciones en las curvas de validación, hemos aumentado el número de ejemplos de validación a 5990, hemos disminuido el *learning rate* a $\eta = 10^{-4}$ y, por último, hemos aumentado el tamaño del *mini-batch* a 30 ejemplos. Los resultados muestran claramente una mejora en la precisión del modelo conforme aumenta su complejidad, saturándose en 5 capas ocultas de 200 neuronas. Podemos visualizar dicha mejora en la gráfica de la derecha en 4.10, donde se comparan las curvas de entrenamiento entre el peor modelo (A, 1 capa de 20 neuronas) y el mejor modelo (B, 5 capas de 200 neuronas).

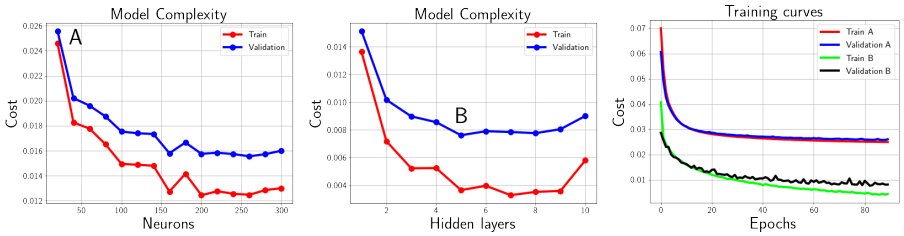

Figura 4.10: Curva de complejidad para el número de neuronas (izquierda), curva de complejidad para número de capas (centro) y curvas de entrenamiento del peor (A) y el mejor (B) modelo (derecha).

Para terminar, mostraremos gráficamente en la figura 4.11 la predicción del potencial sobre un par de ejemplos de espectro $T(k)$ (provenientes de los ejemplos de validación), tanto en el modelo A como en el modelo B. En ellos es posible apreciar de manera sutil el aumento en la precisión del modelo B.

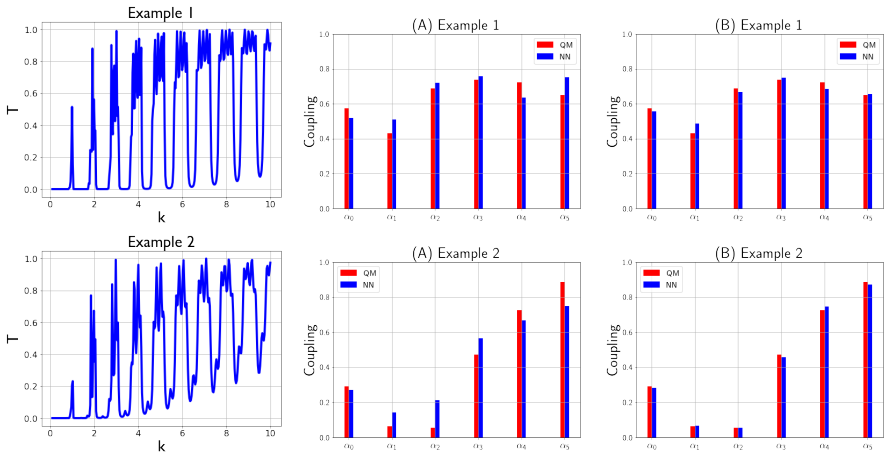

Figura 4.11: Ejemplos de predicción. Los acoplamientos están normalizados entre 0 y 1.

Capítulo 5

Conclusiones

En este trabajo hemos mostrado que los modelos de aprendizaje automático conocidos como redes neuronales artificiales son claramente capaces de aprender a realizar cálculos con precisión que solucionen la ecuación de Schrödinger en la geometría de scattering unidimensional para partículas materiales. En concreto, las hemos entrenado para que calculen espectros de transmisión a partir de un potencial arbitrario. Además, hemos conseguido que también resuelvan el problema inverso: dado un espectro de transmisión, calcular el potencial del cual proviene.

Para asegurarnos de que adquieren esa capacidad de resolución y generalización, hemos dejado que las redes aprendan sobre configuraciones aleatorias de potenciales del modelo de Kronig-Penney, ya que en este caso los espectros de transmisión contienen mucha información acerca de los potenciales.

Además, en los modelos de red directos, hemos observado una mejora considerable en la predicción y en la facilidad de aprendizaje por parte de los modelos que predicen todo el espectro de transmisión en la capa *output* con respecto a los que solo predicen una transmitancia. A pesar de ello, estos últimos también son capaces de realizar predicciones muy precisas, aunque de manera más limitada. Dicha limitación se encuentra principalmente en el número de deltas de Dirac del potencial que reciben como *input*.

Capítulo 6

Bibliografía

[1] M. A. Nielsen. *Neural Networks and Deep Learning*. Determination Press, 2015.

[2] Diu B. Laloë F Cohen-Tannoudji, C. *Quantum Mechanics*. Wiley, 1977.

[3] A. Rodríguez. *Quantum wires in one dimension: disorder, electronic transport and dissipation*. PhD thesis, Universidad de Salamanca, 2005.

[4] R. Kronig and W. Penney. Quantum mechanics of electrons in crystal lattices. *Proceedings of The Royal Society A: Mathematical, Physical and Engineering Sciences*, 130, 1931.

Anexos

Anexos A

Método de la matriz de transferencia

Presentamos en este anexo uno de los métodos numéricos utilizados en este trabajo para el cálculo de transmitancia en función del número de ondas k. Partimos de la ecuación de Schrödinger independiente del tiempo en un sistema de scattering unidimensional, en el que el potencial $V(x)$ es nulo en todo el espacio excepto en la región $x \in [0, L]$:

$$-\frac{\hbar^2}{2m}\frac{d^2\psi}{dx^2} + V(x)\psi(x) = E\psi(x). \tag{A.1}$$

Dado un intervalo espacial a suficientemente pequeño, podemos aproximar la derivada segunda de la función de onda de la siguiente manera:

$$\frac{d^2\psi}{dx^2} \simeq \frac{\psi(x+a) - 2\psi(x) + \psi(x-a)}{a^2}. \tag{A.2}$$

Sustituyendo A.2 en A.1 obtenemos:

$$-[\psi(x+a) - 2\psi(x) + \psi(x-a)] + \frac{2ma^2}{\hbar^2}V(x)\psi(x) = \frac{2ma^2}{\hbar^2}E\psi(x). \tag{A.3}$$

Por simplicidad de las expresiones, adimensionalizamos la energía y el potencial de la siguiente manera: $\tilde{E} = 2ma^2\hbar^{-2}E$ y $\tilde{V}(x) = 2ma^2\hbar^{-2}V(x)$. Así, obtenemos:

$$-\psi(x+a) + 2\psi(x) - \psi(x-a) + \tilde{V}(x)\psi(x) = \tilde{E}\psi(x). \tag{A.4}$$

A partir de ahora nos referiremos a $\tilde{E}$ y $\tilde{V}(x)$ simplemente como E y $V(x)$. Dicho esto, vamos a realizar un discretizado espacial en la región $x \in [0, L]$. Consideramos los puntos $\{x_0, ..., x_N\}$ (con $x_0 = 0$ y $x_N = L$) tales que $x_n = na$, con $n \in \mathbb{Z}$. Podemos expresar tanto el potencial como la función de onda en un punto x_n como V_n y ψ_n respectivamente. Si reescribimos esta notación en A.4, obtenemos:

$$\psi_{n+1} - 2\psi_n + \psi_{n-1} - V_n\psi_n = -E\psi_n \tag{A.5}$$

Esta ecuación la podemos reescribir matricialmente de la sigueinte manera:

$$\begin{pmatrix} \psi_{n+1} \\ \psi_n \end{pmatrix} = \begin{pmatrix} 2 + V_n - E & -1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} \psi_n \\ \psi_{n-1} \end{pmatrix}, \tag{A.6}$$

Observamos en la ecuación A.6 que podemos calcular ψ_{n+1} a partir de ψ_n y ψ_{n-1} utilizando la matriz que definimos como:

$$T_{n+1\leftarrow n} = \begin{pmatrix} 2 + V_n - E & -1 \\ 1 & 0 \end{pmatrix} \tag{A.7}$$

Por tanto, si conocemos ψ_{-1} y ψ_0, podemos llegar a calcular ψ_N y ψ_{N+1} por medio del producto de matrices $T_{n+1\leftarrow n}$, el cual llamamos *matriz de transferencia* $\mathbf{T}$:

$$\begin{pmatrix} \psi_{N+1} \\ \psi_N \end{pmatrix} = \mathbf{T} \begin{pmatrix} \psi_0 \\ \psi_{-1} \end{pmatrix}, \tag{A.8}$$

$$\mathbf{T} = T_{N+1\leftarrow N} T_{N\leftarrow N-1} ... T_{2\leftarrow 1} T_{1\leftarrow 0} = \begin{pmatrix} T_{11} & T_{12} \\ T_{21} & T_{22} \end{pmatrix} \tag{A.9}$$

Consideramos ahora las soluciones de A.1 en la región $x \notin [0, L]$, teniendo en cuenta que la partícula se aproxima desde la izquierda, y que t y r son los coeficientes de transmisión y reflexión. Para $x < 0$, tenemos:

$$\psi(x) = e^{ikx} + re^{-ikx}, \tag{A.10}$$

mientras que en $x \geq L$ tenemos:

$$\psi(x) = te^{ikx}. \tag{A.11}$$

Utilizando las expresiones de arriba y la discretización espacial en $x \in [0, L]$, podemos establecer condiciones de contorno en $x = 0$ y $x = L$:

$$\psi(0) = 1 + r = \psi_0, \tag{A.12}$$

$$\psi(-a) = e^{-ika} + re^{ika} = \psi_{-1}, \tag{A.13}$$

$$\psi(L) = t = \psi_N, \tag{A.14}$$

$$\psi(L + a) = te^{ika} = \psi_{N+1}. \tag{A.15}$$

Ahora, sutituimos las expresiones de A.12 a A.15 en la definición A.8. Obtenemos:

$$\begin{pmatrix} te^{ika} \\ t \end{pmatrix} = \begin{pmatrix} T_{11} & T_{12} \\ T_{21} & T_{22} \end{pmatrix} \begin{pmatrix} 1 + r \\ e^{-ika} + re^{ika} \end{pmatrix} \tag{A.16}$$

De esta ecucación matricial se obtienen los coeficientes de transmisión y reflexión. Y con ellos, la transmitancia $T = |t|^2$:

$$r = \frac{T_{11} + T_{12}e^{-ika} - T_{21}e^{ika} - T_{22}}{-T_{11} + (-T_{12} + T_{21})e^{ika} + T_{22}e^{2ika}} \tag{A.17}$$

$$t = (T_{11}e^{-ika} + T_{12})r + T_{11}e^{-ika} + T_{12}e^{-2ika} \tag{A.18}$$